Journal of Theoretical and Applied Information Technology

www.jatit.org

BACKWARD ERROR RECOVERY PROTOCOLS IN DISTRIBUTED MOBILE SYSTEMS: A SURVEY

[1] Sunil KumarGupta, [2]R. K Chauhan, [3]Parveen Kumar

[1]Asstt Prof., Department of Computer Sc. & Engg., BCET, Gurdaspur-143521, India

[2] Professor, Department of Computer Science, Kurukshetra University, Kurukshetra, India

[3]Professor, Computer Science, APIIT, Panipat, India

[1]E-mail: skgbcet1965@rediffmail.com

ABSTRACT

This survey covers backward error recovery techniques for distributed systems specially the distributed mobile systems. Backward error recovery protocols have been classified into user triggered checkpointing and transparent checkpointing. Transparent checkpointing can be uncoordinated checkpointing, Coordinated checkpointing, Quasi Synchronous or communication-induced checkpointing and Message Logging based Checkpointing. Through out this survey we, highlight the research issues that are at the core of backward error recovery and present the solutions that currently address them.

Keywords: *Distributed mobile system, Co-ordinated checkpointing, Domino effect, Message logging.*

1. INTRODUCTION

A distributed system is a collection of processes that communicate with each other by exchanging messages. A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to co-ordinate their activities and to share the resources of the system, so that users perceive the systems as a single, integrated computing facility. Recent years have witnessed rapid development of mobile communications. In the future, we will expect more and more people will use some portable units such as notebooks or personal data assistants. A mobile distributed computing system is a distributed system where some of the processes are running on mobile hosts (MHs). The term "mobile" implies able to move while retaining its network connections. A host that can move while retaining its network connections is an MH. An MH communicates with other nodes of the system via a special node called mobile support station (MSS) [1], [2], [15], [16]. An MH can directly communicate with an MSS (and vice versa) only if the MH is physically located within the cell serviced by the MSS. A cell is a geographical area around an MSS in which it can support an MH. An MH can change its geographical position freely from one cell to another or even to an area covered by no cell. At any given instant of time, an MH may logically belong to only one cell; its current cell defines the MH's location, and the MH is considered local to the MSS providing wireless coverage in the cell. An MSS has both wired and wireless links and acts as an interface between the static network and a part of the mobile network. Static network connects all MSSs. A static node that has no support to MH can be considered as an MSS with no MH. Critical applications are required to execute fault-tolerantly on such systems [1], [2], [7], [50], [66].

The system model for supporting host mobility consists of two distinct sets of entities: a large number of MHs and relatively fewer numbers of MSSs. All fixed hosts and the communication path between them constitute the static/fixed network. The fixed network connects islands of wireless cells, each comprising of an MSS and the local MHs. The static network provides reliable, sequenced delivery of messages between any two MSSs, with arbitrary message latency. Similarly, the wireless network within a cell ensures FIFO delivery of messages between an MSS and a local MH, i.e., there exists a FIFO channel from an MH to its local MSS, and another FIFO channel from the MSS to the MH. If an MH did not leave the cell, then every message sent to it from the local MSS would be received in the sequence in which they are sent [1], [2], [15], [16], [25].

Message communication from an MH MH_1 to another MH MH_2 occurs as follows. MH_1 first sends the message to its local MSS MSS_1 using the wireless link. MSS_1 forwards it to MSS_2, the local MSS of MH_2, via the fixed network.

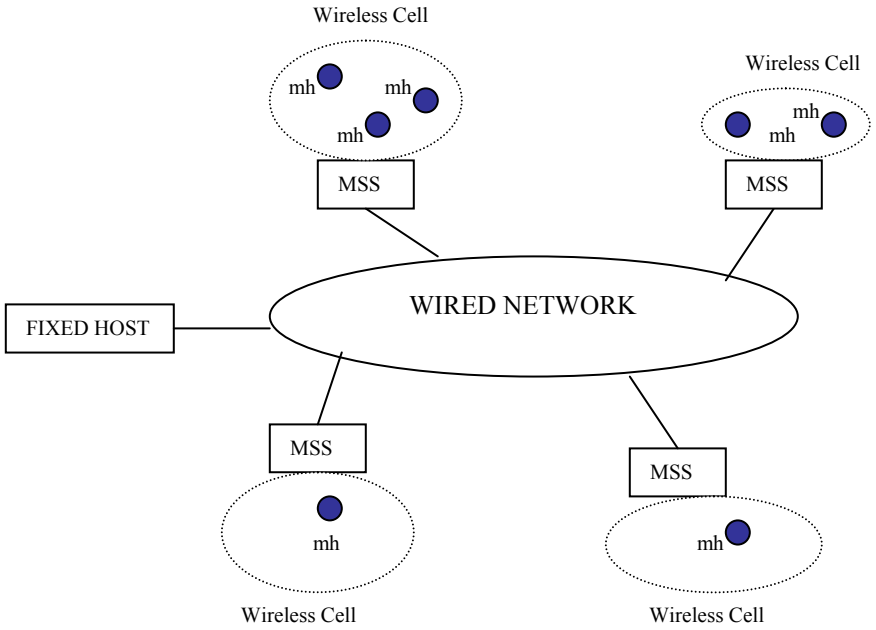

Figure 1.1 The system model for supporting host mobility

MSS_2 then transmits it to MH_2 over its wireless network. However, the location of MH_2 may not be known to MSS_1, therefore, MSS_1 may require to first determining the location of MH_2. This is essentially the problem faced by network layer routing protocols [13], [30], [57], [62].

Mobile Hosts often disconnect from the rest of the network. In our model, disconnection is distinct from failure. Disconnections are elective or volunteer by nature, so a mobile host informs the system prior to its occurrence and executes an application-specific disconnection protocol if necessary [2]. Disconnection can be voluntary on involuntary [35]. We use the term "disconnection" to always imply a voluntary disconnection. We refer to an abrupt or involuntary disconnection as a failure.

2. BACKWARD ERROR RECOVERY

Fault tolerance can be achieved through some kind of redundancy. Redundancy can be temporal or spatial. In temporal redundancy, i.e., checkpoint-restart, an application is restarted from an earlier checkpoint or recovery point after a fault. This may result in the loss of some processing and applications may not be able to meet strict timing targets. In spatial redundancy, many copies of the application execute on different processors concurrently and strict timing constraints can be met. But the cost of providing fault tolerance using spatial redundancy is quite high and may require extra hardware. Checkpoint-Restart or Backward error recovery is quite inexpensive and does not require extra hardware in general. Besides providing fault tolerance, checkpointing can be used for process migration, debugging distributed applications, job swapping, postmortem analysis and stable property detection [63].

There are two approaches for error recovery:

- Forward Error Recovery
- Backward Error Recovery

In forward error recovery techniques, the nature of errors and damage caused by faults must be completely and accurately assessed and so it becomes possible to remove those errors in the process state and enable the process to move forward [46]. In distributed system, accurate assessment of all the faults may not be possible.

In backward error recovery techniques, the nature of faults need not be predicted and in case of error, the process state is restored to previous error-free state. It is independent of the

nature of faults. Thus, backward error recovery is more general recovery mechanism [14], [42].

There are three steps involved in backward-error recovery. These are:

- Checkpointing the error-free state periodically
- Restoration in case of failure
- Restart from the restored state

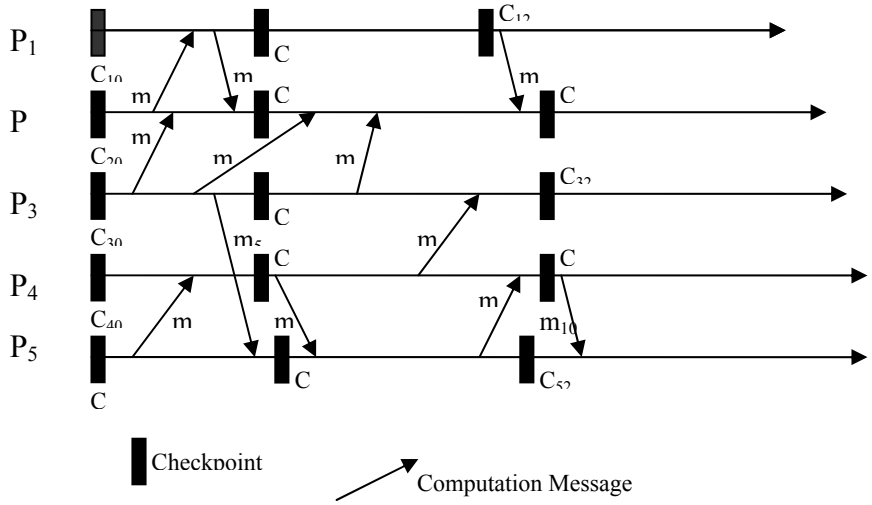

Figure 2.1 Consistent and Inconsistent

Backward error recovery is also known as checkpoint-restore-restart (CRR) or checkpoint-restart (CR). The checkpointing process is executed periodically to advance the recovery line.

A checkpoint is a local state of a process saved on stable storage. In a distributed system, since the processes in the system do not share memory, a global state of the system is defined as a set of local states, one from each process. The state of channels corresponding to a global state is the set of messages sent but not yet received. A lost or in-transit message is one, the sending of which has been recorded by the sender but whose receiving could not be recorded by the receiving process. An orphan message is a message whose receive event is recorded, but its send event is lost. A global state is said to be "consistent" if it contains no orphan message and all the in-transit messages are logged. In Figure 2.1, the initial global state $\{C_{10}, C_{20}, C_{30}, C_{40}, C_{50}\}$ is consistent. It should be noted that initial global state is always consistent, because, it can not contain any orphan message. The Global State $\{C_{11}, C_{21}, C_{31}, C_{41}, C_{51}\}$ is also consistent, because, it does not possess any orphan message. It needs to be noted that by definition, m_0 is not an orphan message but in-transit message. The Global State $\{C_{12}, C_{22}, C_{32}, C_{42}, C_{52}\}$ is inconsistent because it includes the orphan message m_8. By definition, m_8 is an orphan message. To recover from a failure, the system restarts its execution from a previous consistent global state saved on the stable storage during fault-free execution. This saves all the computation done up to the last checkpointed state and only the computation done thereafter needs to be redone [8], [51], [52].

After a failure, a system must be restored to a consistent system state. Essentially, a system state is consistent if it could have occurred during the preceding execution of the system from its initial state, regardless of the relative speeds of individual processes. This assumes that the total execution of the system is equivalent to some fault free execution [8]. It has been shown that two local checkpoints being causally unrelated is a necessary but not sufficient condition for them to belong to the same consistent global checkpoint. This problem was first addressed by Netzer and Xu who introduced the notion of a Z-path between local checkpoints to capture both their causal and hidden dependencies [44]. Considering a checkpoint and communication pattern, the rollback dependency trackability property stipulates that there is no hidden dependency between local checkpoints [11]. To be able to recover a system state, all of its individual process states must be able to be restored. A consistent system state in which each process state can be restored is thus called a recoverable system state.

Processes in a distributed system communicate by sending and receiving messages. A process can record its own state and messages it sends and receives; it can record nothing else. To determine a global system state, a process P_i must enlist the cooperation of other processes that must record their own local states and send the recorded local states to P_i. All processes cannot record their local states at precisely the same instant unless they have access to a common clock. We assume that processes do not share clocks or memory. The problem is to devise algorithms by which processes record their own states and the states of communication channels so that the set of process and channel states recorded form a global system state. The global state detection algorithm is to be superimposed on the underlying computation; it must run concurrently with, but not alter, this underlying computation [19].

The state detection algorithm plays the role of a group of photographers observing a panoramic, dynamic scene, such as a sky filled with migrating birds- a scene so vast that it can not be captured by a single photograph. The photographers must take several snapshots and piece the snapshots together to form a picture of the overall scene. All snapshots cannot be taken at precisely the same instant because of synchronization problems. Furthermore, the photographers should not disturb the process that is being photographed. Yet, the composite picture should be meaningful. The problem before us is to define meaningful and then to determine how the photographs should be taken [19].

The problem of taking a checkpoint in a message passing distributed system is quite complex because any arbitrary set of checkpoints cannot be used for recovery [19], [51], [52]. This is due to the fact that the set of checkpoints used for recovery must form a consistent global state.

In backward error recovery, depending on the programmer's intervention in process of checkpointing, the classification can be:

- User-Triggered checkpointing
- Transparent Checkpointing

User triggered checkpointing schemes require user interaction and are useful in reducing the stable storage requirement [20]. These are generally employed where the user has the knowledge of the computation being performed and can decide the location of the checkpoints. The main problem is the identification of the checkpoint location by a user.

The **transparent checkpointing** techniques do not require user interaction and can be classified into following categories:

- Uncoordinated Checkpointing
- Coordinated Checkpointing
- Quasi-Synchronous or Communication-induced Checkpointing
- Message Logging based Checkpointing

2.1 UNCOORDINATED CHECKPOINTING

In uncoordinated or independent checkpointing, processes do not coordinate their checkpointing activity and each process records its local checkpoint independently [14], [54], [64].

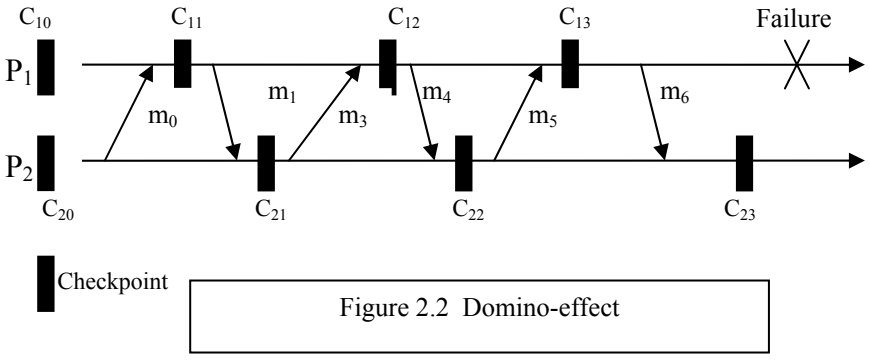

Figure 2.2 Domino-effect

It allows each process the maximum autonomy in deciding when to take checkpoint, i.e., each process may take a checkpoint when it is most convenient. It eliminates coordination overhead all together and forms a consistent global state on recovery after a fault [14]. After a failure, a consistent global checkpoint is established by tracking the dependencies. It may require cascaded rollbacks that may lead to the initial state due to domino-effect [36], [51], [52]. It requires multiple checkpoints to be saved for each process and

periodically invokes garbage collection algorithm to reclaim the checkpoints that are no longer needed. In this scheme, a process may take a useless checkpoint that will never be a part of global consistent state. Useless checkpoints incur overhead without advancing the recovery line [20].

The main disadvantage of this approach is the domino-effect [Figure 2.2]. In this example, processes P_1 and P_2 have independently taken a sequence of checkpoints. The interleaving of messages and checkpoints leave no consistent set of checkpoints for P_1 and P_2, except the initial one at $\{C_{10}, C_{20}\}$. Consequently, after P_1 fails, both P_1 and P_2 must roll back to the beginning of the computation [36]. It should be noted that global state $\{C_{11}, C_{21}\}$ is inconsistent due to orphan message m_1. Similarly, global state $\{C_{12}, C_{22}\}$ is inconsistent due to orphan message m_4.

2.2 COORDINATED CHECKPOINTING

In coordinated or synchronous checkpointing, processes take checkpoints in such a manner that the resulting global state is consistent. Mostly it follows two-phase commit structure [19], [21], [36]. In the first phase, processes take tentative checkpoints and in the second phase, these are made permanent. The main advantage is that only one permanent checkpoint and at most one tentative checkpoint is required to be stored. In case of a fault, processes rollback to last checkpointed state. A permanent checkpoint can not be undone. It guarantees that the computation needed to reach the checkpointed state will not be repeated. A tentative checkpoint, however, can be undone or changed to be a permanent checkpoint.

A straightforward approach to coordinated checkpointing is to block communications while the checkpointing protocol executes [56]. A coordinator takes a checkpoint and broadcasts a request message to all processes, asking them to take a checkpoint. When a process receives the message, it stops its executions, flushes all the communication channels, takes a tentative checkpoint, and sends an acknowledgement message back to the coordinator. After the coordinator receives acknowledgements from all processes, it broadcasts a commit message that completes the two-phase checkpoint protocol. On receiving commit, a process converts its tentative checkpoint into permanent one and discards its old permanent checkpoint, if any. The process is then free to resume execution and exchange messages with other processes.

The coordinated checkpointing protocols can be classified into two types: blocking and non-blocking. In blocking algorithms, as mentioned above, some blocking of processes takes place during checkpointing [36], [56]. In non-blocking algorithms, no blocking of processes is required for checkpointing [19], [21]. The coordinated checkpointing algorithms can also be classified into following two categories: minimum-process and all process algorithms. In all-process coordinated checkpointing algorithms, every process is required to take its checkpoint in an initiation [19], [21]. In minimum-process algorithms, minimum interacting processes are required to take their checkpoints in an initiation [36].

2.3 QUASI-SYNCHRONOUS OR COMMUNICATION INDUCED CHECKPOINTING

Communication-induced checkpointing avoids the domino-effect without requiring all checkpoints to be coordinated [12], [26], [41]. In these protocols, processes take two kinds of checkpoints, local and forced. Local checkpoints can be taken independently, while forced checkpoints are taken to guarantee the eventual progress of the recovery line and to minimize useless checkpoints. As opposed to coordinated checkpointing, these protocols do no exchange any special coordination messages to determine when forced checkpoints should be taken. But, they piggyback protocol specific information [generally checkpoint sequence numbers] on each application message; the receiver then uses this information to decide if it should take a forced checkpoint. This decision is based on the receiver determining if past communication and checkpoint patterns can lead to the creation of useless checkpoints; a forced checkpoint is taken to break these patterns [20], [41].

2.4 MESSAGE LOGGING BASED CHECKPOINTING PROTOCOLS

Message-logging protocols (for example [3], [4], [5], [6], [9], [22], [23], [33], [49], [55], [58], [59], [60], [61]), are popular for building systems that can tolerate process crash failures. Message logging and checkpointing can be used to provide fault tolerance in distributed systems in which all inter-process communication is through messages. Each message received by a process is saved in message log on stable storage. No coordination is required between the checkpointing of different processes or between

message logging and checkpointing. The execution of each process is assumed to be deterministic between received messages, and all processes are assumed to execute on fail stop processes.

When a process crashes, a new process is created in its place. The new process is given the appropriate recorded local state, and then the logged messages are replayed in the order the process originally received them. All message-logging protocols require that once a crashed process recovers, its state needs to be consistent with the states of the other processes [20], [65]. This consistency requirement is usually expressed in terms of orphan processes, which are surviving processes whose states are inconsistent with the recovered states of crashed processes. Thus, message- logging protocols guarantee that upon recovery, no process is an orphan. This requirement can be enforced either by avoiding the creation of orphans during an execution, as pessimistic protocols do, or by taking appropriate actions during recovery to eliminate all orphans as optimistic protocols do. Bin Yao et al. [65] describes a receiver-based message logging protocol for mobile hosts, mobile support stations and home agents in a Mobile IP environment, which guarantees independent recovery. Checkpointing is utilized to limit log size and recovery latency.

3. CHECKPOINTING ISSUES IN DISTRIBUTED MOBILE SYSTEMS

The existence of mobile nodes in a distributed system introduces new issues that need proper handling while designing a checkpointing algorithm for such systems. These issues are mobility, disconnections, finite power source, vulnerable to physical damage, lack of stable storage etc. [1], [10]. The location of an MH within the network, as represented by its current local MSS, changes with time. Checkpointing schemes that send control messages to MHs, will need to first locate the MH within the network, and thereby incur a search overhead [2]. Due to vulnerability of mobile computers to catastrophic failures, disk storage of an MH is not acceptably stable for storing message logs or local checkpoints. Checkpointing schemes must therefore, rely on an alternative stable repository for an MH's local checkpoint [2]. Disconnections of one or more MHs should not prevent recording the global state of an application executing on MHs. It should be noted that disconnection of an MH is a voluntary operation, and frequent disconnections of MHs is an expected feature of the mobile computing environments [2]. The battery at the MH has limited life. To save energy, the MH can power down individual components during periods of low activity [24]. This strategy is referred to as the doze mode operation. The MH in doze mode is awakened on receiving a message. Therefore, energy conservation and low bandwidth constraints require the checkpointing algorithms to minimize the number of synchronization messages and the number of checkpoints.

The new issues make traditional checkpointing techniques unsuitable to checkpoint mobile distributed systems [1], [18], [43], [48]. Prakash-Singhal [48] proposed that a good checkpointing protocol for mobile distributed systems should have low memory overheads on MHs, low overheads on wireless channels and should avoid awakening of an MH in doze mode operation. The disconnection of MHs should not lead to infinite wait state. The algorithm should be non-intrusive, coordinated, and should force minimum number of processes to take their local checkpoints.

Minimum-process coordinated checkpointing is an attractive approach to introduce fault tolerance in mobile distributed systems transparently. It avoids domino-effect, minimizes stable storage requirements, and forces only minimum interacting processes to checkpoint. To recover from a failure, the system simply restarts its execution from a previous consistent global checkpoint saved on the stable storage. But, it has the following disadvantages. Some blocking of processes takes place or some useless checkpoints are taken. In order to record a consistent global checkpoint, processes must synchronize their checkpointing activities. In other words, when a process initiates checkpointing procedure, it asks all relevant processes to take their checkpoints. Therefore, coordinated checkpointing suffers from high overhead associated with the checkpointing process. Sometimes, checkpoint sequence numbers are piggybacked along with computation messages. If a single process fails to checkpoint, the whole checkpointing effort of the particular initiation goes waste.

Acharya, A. [2] cast distributed systems with mobile hosts into a two tier structure: 1) a network of fixed hosts with more resources in terms of storage, computing, and communication, and 2) mobile hosts, which may operate in a disconnected, or doze mode, connected by a low bandwidth wireless connection to this network. They propose a two-tier principle for structuring distributed algorithms for this model:

To the extent possible, computation and communication costs of an algorithm is borne by the static network. The core objective of the algorithm is achieved through a distributed execution amongst the fixed hosts while performing only those operations at the mobile hosts that are necessary for the desired overall functionality.

In wireless cellular network, mobile computing based on a two-tier coordinated checkpointing algorithm reduces the number of synchronization messages [37].

4. PRELIMINARIES

When processes interact with each other by exchanging messages, dependency is introduced among the events of different processes, making it difficult to have a total ordering of events. Lamport [40] pointed out this and he proposed a relation called 'happened before' (denoted by →) to have a partial ordering of events in a distributed system. This is an irreflexive, anti-symmetric, transitive relation.

If a and b are two events occurring in the same process and if a occurs before b, then a→b. If a is the event of sending a message and b is the event of receiving the same message, then a→b. Two events a and b are said to be concurrent if and only if a does not happen before b and b does not happen before a. Local checkpoint is an event that records the state of a process at a processor at a given instant. Global checkpoint is a collection of local checkpoints, one from each process. A global state is said to be consistent if all the included events form a concurrent set. A consistent global checkpoint is a collection of local checkpoints, one from each process, such that each local checkpoint is concurrent to every other local checkpoint. Rollback recovery is a process of resuming/recovering a computation from a consistent global checkpoint.

The messages generated by the underlying computation are referred to as computation messages or simply messages and are denoted by m_i or m. The processes are denoted by P_i. The i^{th} CI of a process denotes all the computation performed between its i^{th} and $(i+1)^{th}$ checkpoint, including the i^{th} checkpoint but not the $(i+1)^{th}$ checkpoint.

A process P_i directly depends upon P_j only if there exist m such that: (i) P_i has processed m sent by P_j (ii) P_i has not taken any permanent checkpoint after processing m (iii) P_j has not taken any permanent checkpoint after sending m. Direct dependencies at P_i can be stored in a bit vector of length n for n processes [say $ddv_i[]$]. $ddv_i[j]=1$ implies P_i is directly dependent upon P_j. In minimum-process coordinated checkpointing, if P_i takes its checkpoint and P_i is dependent upon P_j, then P_j should also take its checkpoint. Minimum set is the set of processes, which need to checkpoint in an initiation. A process is in the minimum set only if the initiator process is transitively dependent upon it. A process that initiates checkpointing is called initiator process or simply initiator. The minimum-process algorithms are generally based on keeping track of direct dependencies among processes and computing minimum set [38], [45].

Once the system has rolled back to a consistent state, the nodes have to retrace their computation that was undone during the rollback. The following types of messages have to be handled while retracing the lost computation [48].

- Orphan Messages: Messages whose reception has been recorded, but the record of their transmission has been lost. This situation arises when the sender node rolls back to a state prior to sending the message while the receiver node still has the record of its reception.
- Lost Messages: Messages whose transmission has been recorded, but the record of their reception has been lost. This happens if the receiver rolls back to a state prior to the reception of the message, while the sender does not roll back to a state prior to their sending.
- Duplicate Messages: This happens when more than one copy of the same message arrives at a node; perhaps one corresponding to the original computation and one generated during recovery phase. If the first copy has been processed, all subsequent copies should be discarded.

  In deterministic systems, if two processes start in the same state, and both receive the identical sequence of inputs, they will produce the identical sequence outputs and will finish in the same state. The state of a process is thus completely determined by its starting state and by sequence of messages it has received [31], [32].

Chandy-Lamport algorithm [19] works with FIFO channels only. If a message m_1 followed by m_2 is sent from P_i to P_j, m_1 reaches before m_2 when the channels are FIFO. Advantage of a FIFO channel is that without explicitly sending any message sequence numbers with messages, it is possible to arrange the messages in a sequence. Non-FIFO channels necessitate headers with regular messages to ensure correct ordering of messages [53]. Headers should

contain sequence numbers of regular messages. The possibility of non-FIFO channel is justified in a distributed environment, since it is possible for messages to be routed through different channels and reach the destination out of order.

In a centralized algorithm like Chandy-lamport [19], there is one node, which always initiates the checkpoints and coordinates the participating nodes. The disadvantage of a centralized algorithm is that all nodes have to initiate checkpoints whenever the centralized node decides to checkpoint. Nodes can be given autonomy in initiating checkpoints by allowing any node in the system to initiate checkpoints. Such a distributed checkpointing algorithm can initiate complete checkpointing [39] or selective checkpointing [36].

5. CONCLUSION

We have reviewed different approaches to rollback recovery with respect to a set of properties including performance overhead, storage over-head, ease of recovery, freedom from domino effect, freedom from orphan processes, and the extent of rollback. Checkpointing protocols require the processes to take periodic checkpoints with varying degrees of coordination. Coordinated checkpointing requires the processes to coordinate their checkpoints to form global consistent system states. Coordinated checkpointing generally simplifies recovery and garbage collection, and yields good performance in practice. At the other end of the spectrum, uncoordinated checkpointing does not require the processes to coordinate their checkpoints, but it suffers from potential domino effect, complicates recovery, and still requires coordination to perform output commit or garbage collection. Between these two ends are communication-induced checkpointing schemes that depend on the communication patterns of the applications to trigger checkpoints. These schemes do not suffer from the domino effect and do not require coordination. Recent studies, however, have shown that the non-deterministic nature of these protocols complicates garbage collection and degrades performance. Log-based rollback recovery is often a natural choice for applications that frequently interact with the outside world. It allows efficient output commit, and has three flavors, pessimistic, optimistic, and causal. This form of logging simplifies recovery, output commit, and protects surviving processes from having to roll back. These advantages have made pessimistic logging attractive in commercial environment where simplicity and robustness are necessary. Causal logging reduces the overhead while still preserving the properties of fast output commit and orphan-free recovery.

REFERENCES

[1] Acharya A. and Badrinath B. R., "Checkpointing Distributed Applications on Mobile Computers," *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems*, pp. 73-80, September 1994.

[2] Acharya A., "Structuring Distributed Algorithms and Services for networks with Mobile Hosts", *Ph.D. Thesis*, *Rutgers University*, 1995.

[3] Alvisi, Lorenzo and Marzullo, Keith," Message Logging: Pessimistic, Optimistic, Causal, and Optimal", *IEEE Transactions on Software Engineering*, Vol. 24, No. 2, February 1998, pp. 149-159.

[4] L. Alvisi, Hoppe, B., Marzullo, K., "Nonblocking and Orphan-Free message Logging Protocol," *Proc. of 23rd Fault Tolerant Computing Symp.*, pp. 145-154, June 1993.

[5] L. Alvisi," Understanding the Message Logging Paradigm for Masking Process Crashes," *Ph.D. Thesis, Cornell Univ., Dept. of Computer Science*, Jan. 1996. Available as Technical Report TR-96-1577.

[6] L. Alvisi and K. Marzullo," Tradeoffs in implementing Optimal Message Logging Protocol", *Proc. 15th Symp. Principles of Distributed Computing*, pp. 58-67, ACM, June, 1996.

[7] Adnan Agbaria, Wiilliam H Sanders," Distributed Snapshots for Mobile Computing Systems", *IEEE Intl. Conf. PERCOM'04*, pp. 1-10, 2004.

[8] Avi Ziv and Jehoshua Bruck, " Checkpointing in Parallel and Distributed Systems", *Book Chapter from Parallel and Distributed Computing Handbook edited by Albert Z. H. Zomaya*, pp. 274-302, Mc Graw Hill, 1996.

[9] A. Borg, J. Baumbach, and S. Glazer," A

Message System Supporting Fault Tolerance", *Proc. Symp. Operating System Principles*, pp. 90-99, ACM SIG OPS, Oct. 1983.

[10] Adnan Agbaria, William H. Sanders, " Distributed Snapshots for Mobile Computing Systems", *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (Percom'04),* pp. 1-10, 2004.

[11] Baldoni R., Hélary J-M., Mostefaoui A. and Raynal M., " Rollback Dependency Trackability: A Minimial Characterization and its Protocol", *Information and Computation*, 165, pp. 144-173, 2003.

[12] Baldoni R., Hélary J-M., Mostefaoui A. and Raynal M., "A Communication-Induced Checkpointing Protocol that Ensures Rollback-Dependency Trackability," *Proceedings of the International Symposium on Fault-Tolerant-Computing Systems*, pp. 68-77, June 1997.

[13] Bhagwat P., and Perkins, C.E., "A mobile Networking System based on Internet Protocol (IP)",*USENIX Symposium on Mobile and Location-Independent Computing,* August 1993.

[14] Bhargava B. and Lian S. R., "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems-An Optimistic Approach," *Proceedings of 17th IEEE Symposium on Reliable Distributed Systems*, pp. 3-12, 1988.

[15] Badrinath B. R, Acharya A., T. Imielinski "Structuring Distributed Algorithms for Mobile Hosts", *Proc. 14th Int. Conf. Distributed Computing Systems*, June 1994.

[16] Badrinath B. R, Acharya A., T. Imielinski " Designing Distributed Algorithms for Mobile Computing Networks", *Computer Communications*, Vol. 19, No. 4, 1996.

[17] Cao G. and Singhal M., "On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," *Proceedings of International Conference on Parallel Processing*, pp. 37-44, August 1998.

[18] Cao G. and Singhal M., "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems," *IEEE Transaction On Parallel and Distributed Systems*, vol. 12, no. 2, pp. 157-172, February 2001.

[19] Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems," *ACM Transaction on Computing Systems*, vol. 3, No. 1, pp. 63-75, February 1985.

[20] Elnozahy E.N., Alvisi L., Wang Y.M. and Johnson D.B., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375-408, 2002.

[21] Elnozahy E.N., Johnson D.B. and Zwaenepoel W., "The Performance of Consistent Checkpointing," Proceedings of the 11th Symposium on Reliable Distributed Systems, pp. 39-47, October 1992.

[22] Elnozahy and Zwaenepoel W, " Manetho: Transparent Roll-back Recovery with Low-overhead, Limited Rollback and Fast Output Commit," *IEEE Trans. Computers*, vol. 41, no. 5, pp. 526-531, May 1992.

[23] Elnozahy and Zwaenepoel W, " On the Use and Implementation of Message Logging," *24th Int'l Symp. Fault Tolerant Computing*, pp. 298-307, IEEE Computer Society, June 1994.

[24] George H. Forman and John Zahorjan, "The Challenges of Mobile Computing", *IEEE Computers* vol. 27, no. 4, pp. 38-47, April 1994.

[25] Richard C. Gass and Bidyut Gupta," An Efficient Checkpointing Scheme for Mobile Computing Systems", *European Simulation Symposium*, Oct 18-20, 2001, pp. 1-6.

[26] Hélary J. M., Mostefaoui A. and Raynal M., "Communication-Induced Determination of Consistent Snapshots,"

Proceedings of the 28th International Symposium on Fault-Tolerant Computing, pp. 208-217, June 1998.

[27] Higaki H. and Takizawa M., "Checkpoint-recovery Protocol for Reliable Mobile Systems," *Trans. of Information processing Japan*, vol. 40, no.1, pp. 236-244, Jan. 1999.

[28] Higaki H. and Takizawa M., "Recovery Protocol for Mobile Checkpointing", *IEEE 9th International Conference on Database Expert Systems Applications, Vienna*, pp. 520-525, 1998

[29] Higaki H. and Takizawa M., "Checkpoint Recovery Protocol for Reliable Mobile Systems", *17th Symposium on Reliable Distributed Systems*, pp. 93-99, Oct. 1998.

[30] Ioannidis, J., Duchamp, D., and Maguire, G.Q., "IP-based protocols for Mobile Internetworking", *In Proc. of ACM SIGCOMM Symposium on Communications, Architectures, and Protocols*, pp. 235-245, September 1991.

[31] Johnson, D.B., Zwaenepoel, W., "Sender-based message logging", *In Proceedingss of 17th international Symposium on Fault-Tolerant Computing*, pp 14-19, 1987.

[32] Johnson, D.B., Zwaenepoel, W., "Recovery in Distributed Systems using optimistic message logging and checkpointing. *In 7th ACM Symposium on Principles of Distributed Computing,* pp 171-181, 1988.

[33] D. Johnson, "Distributed System Fault Tolerance Using Message Logging and Checkpointing," *Ph.D. Thesis, Rice Univ*., Dec. 1989.

[34] JinHo Ahn, Sung-Gi Min, Chong-Sun Hwang, "A Causal Message Logging Protocol for Mobile Nodes in Mobile Computing Environments", *Future Generation Computer Systems* 20, pp 663-686, 2004.

[35] Kistler, J., and Satyanaranyan, M., " Disconnected Operation in the Coda file system", *ACM Trans. on Computer Systems* Vol.10, No.1, February 1992.

[36] Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems," *IEEE Trans. on Software Engineering*, vol. 13, no. 1, pp. 23-31, January 1987.

[37] Kyne-Sup BYUN, Sung_Hwa LIM, Jai-Hoon KIM," Two-Tier Checkpointing Algorithm Using MSS in Wireless Networks", *IEICE Trans. Communications*, Vol E86-B, No. 7, pp. 2136-2142, July 2003.

[38] L. Kumar, M. Misra, R.C. Joshi, "Low overhead optimal checkpointing for mobile distributed systems" *Proceedings. 19th IEEE International Conference on Data Engineering*, pp 686 – 88, 2003.

[39] T.H. Lai and T.H. Yang," On Distributed Snapshots", *Information Processing Letters*, vol. 25, pp. 153-158, 1987.

[40] L. Lamport, "Time, clocks and ordering of events in a distributed system" *Comm. ACM,* vol.21, No.7, pp. 558-565, July 1978.

[41] Manivannan D. and Singhal M., "Quasi-Synchronous Checkpointing: Models, Characterization, and Classification," *IEEE Trans. Parallel and Distributed Systems*, vol. 10, No. 7, pp. 703-713, July 1999.

[42] Manivannan D., Netzer R. H. and Singhal M., "Finding Consistent Global Checkpoints in a Distributed Computation," *IEEE Transactions on Parallel & Distributed Systems*, vol. 8, no. 6, pp. 623-627, June 1997.

[43] Yoshifumi Manabe," A Distributed Consistent Global Checkpoint Algorithm for Distributed Mobile Systems", *8th Int'l Conference on Parallel and Distributed Systems"*, pp. 125-132, 2001.

[44] Netzer, R.H. and Xu,J ,"Necessary and Sufficient Conditions for Consistent Global Snapshots", *IEEE Trans. Parallel and Distributed Systems Vol.* 6, No.2, pp 165-169, 1995.

[45] Parveen Kumar, Lalit Kumar, R K Chauhan, V K Gupta "A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems" *Proceedings of IEEE ICPWC-2005*, January 2005.

[46] Pradhan D.K. and Vaidya N., "Roll-forward Checkpointing Scheme: Concurrent Retry with Non-dedicated Spares," *Proceedings of the IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pp. 166-174, July 1992.

[47] Pushpendra Singh, Gilbert Cabillic, "A Checkpointing Algorithm for Mobile Computing Environment", *LNCS, No. 2775*, pp 65-74, 2003.

[48] Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," *IEEE Transaction On Parallel and Distributed Systems,* vol. 7, no. 10, pp. 1035-1048, October1996.

[49] M.L. Powell and D.L. Presotto, "Publishing: A Reliable Broad case Communication Mechanism", *Proc. ninth Symp. Operating System Principles*, pp. 100-109, ACM SIGOPS, Oct. 1983.

[50] Quagila, F., Ciciani, R., Baldoni, R., " Checkpointing Protocols in Distributed Systems with Mobile Hosts: A Performance Analysis", *IPPS/SPDP Workshop*, pp. 742-755, 1998.

[51] Randall, B, " System Structure for Software Fault Tolerance", *IEEE Trans. on Software Engineering,* Vol.1,No.2, pp 220-232, 1975.

[52] Russell, D.L., "State Restoration in Systems of Communicating Processes", *IEEE Trans. Software Engineering*, Vol.6,No.2, pp 183-194, 1980.

[53] Silva, L.M. and J.G. Silva, "Global checkpointing for distributed programs", *Proc. 11th symp. Reliable Distributed Systems*, pp. 155-62, Oct. 1992.

[54] Storm R., and Temini, S., "Optimistic Recovery in Distributed Systems", *ACM Trans. Computer Systems*, Aug, 1985, pp. 204-226.

[55] A.P. Sistla and J.L. Welch," Efficient Distributed Recovery Using Message Logging", *Proc. 18th Symp. Principles of Distributed Computing*", pp 223-238, Aug. 1989.

[56] Tamir, Y., Sequin, C.H., "Error Recovery in multi-computers using global checkpoints", *In Proceedings of the International Conference on Parallel Processing,* pp. 32-41, 1984.

[57] Terakota, F., Yokote, Y., and Tokoro, M., "A Network Architecture providing host migration transparency", *Proc, of ACM SIGCOMM 91,* September 1991.

[58] S. Venketasan and T.Y. Juang, "Efficient Algorithms for Optimistic Crash recovery", *Distributed Computing*, vol. 8, no. 2, pp. 105-114, June 1994.

[59] S. Venketasan, "Message-Optimal Incremental Snapshots", *Computer and Software Engineering*, vol.1, no.3, pp. 211-231, 1993.

[60] S. Venketasan, " Optimistic Crash recovery Without Rolling back Non-Faulty Processors", *Information Sciences*, 1993.

[61] S. Venketasan and T.T.Y. Juang, "Low Overhead optimistic crash Recovery", *Proc. 11 Int. Conf. Distributed Computing systems*, pp. 454-461, 1991.

[62] Wada H., Yozawa, T., Ohnishi, T. and Tanaka, Y., "Mobile Computing Environment based on internet packet forwarding", *Winter Usenix*, Jan. 1993.

[63] Wang Y. M., Huang Y., Vo K.P., Chung P.Y. and Kintala C., "Checkpointing and its Applications," *Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS-25),* pp. 22-31, June 1995.

[64] Wood, W.G., "A Decentralized Recovery Control Protocol", *IEEE Symposium on Fault Tolerant Computing,* 1981.

[65] Bin Yao, Kuo-Feng Ssu & W. Kect Fuchs, "Message Logging in Mobile Computing", *Proceedings of international conference on FTCS*, pp 294-301, 1999.

[66] Yasuro Sato, Michiko Inoue, Toshimitsu Masuzawa, Hideo Fujiwara, " A Snapshot Algorithm for Distributed Mobile Systems" *Proceedings of the 16th ICDCS*, pp734-743,1996.